Politecnico di Torino - Laurea Magistrale in Ingegneria Informatica
percorso Data Science

Tesina per il corso di Data Spaces
codice corso: 01RLPOV

Docente del corso: Francesco Vaccarino

Studente: Domenico Cefalo - s267569

Anno Accademico 2019/2020

1. Introduzione

1.1 Descrizione dataset e strumenti utilizzati

This data set contains **416 liver patient** records and **167 non liver** patient records.The data set was collected from test samples in North East of Andhra Pradesh, India. '*is_patient*' is a class label used to divide into groups (liver patient or not). Any patient whose age exceeded 89 is listed as being of age "90".

- **age:** *age of the patient;*
- **gender:** *gender of the patient;*
- **tot_bilirubin:** *total bilirubin;*
- **direct_bilirubin:** *direct bilirubin;*
- **alkphos:** *alkaline phoshotase;*
- **sgpt:** *calamine aminotransferase;*
- **sgot:** *aspartate aminotransferase;*
- **tot_proteins:** *total protiens;*
- **albumin:** *albumin;*
- **ag_ratio:** *albumin and globulin ratio;*
- **is_patient:** *selector field used to split the data into two sets;*

For the exploration of data and for all subsequent classification analysis, we decided to use the Python language programming because it is more intuitive and complete of libraries. Mainly we have esed **sklearn**, a very important library in the field of Machine Learning and in data analysis which includes the functions that are used for the preprocessing, reduction of dimensionality, aggregative clustering (to identify the correlation of features) and also for classification algorithms. To better manipulate the data **pandas e numpy library** were used, then for the construction of mathematical plots, reference was made to the **matplotlib library**.

1.2 Data exploration

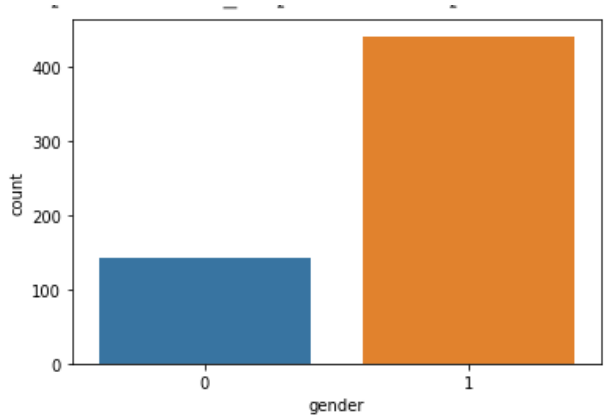
Using pandas, a very useful function is the description that gives us an idea of how numeric values are distributed within the data set. What we find is the following table:

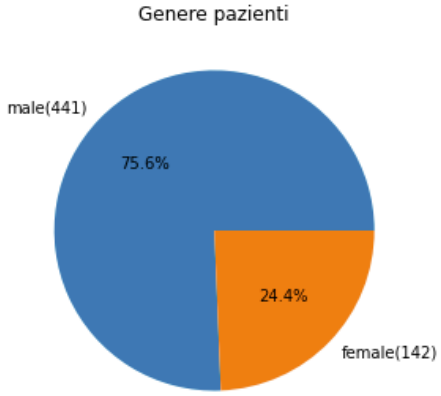
	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	is_patient
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	579.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852	0.947064	1.286449
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519	0.319592	0.452490
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	0.300000	1.000000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000	0.700000	1.000000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	0.930000	1.000000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000	1.100000	2.000000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	2.800000	2.000000

The measure of **count** not is very important for our analysis. Other measure are **mean** that indicates the means for each feature; **std** indicates the standard deviation of each group and which gives us a degrees of data dispersion; **25% (is also called the first quartile), 50% (is generally the median) and 75% (is also called the third quartile); max** indicates the maximum value that each attribute can assume.
We can observe that the scale of values has many differences from each other, for example *direct_bilirubin* has means egual to 0.1 while *tot_proteins* has means egual to 63.

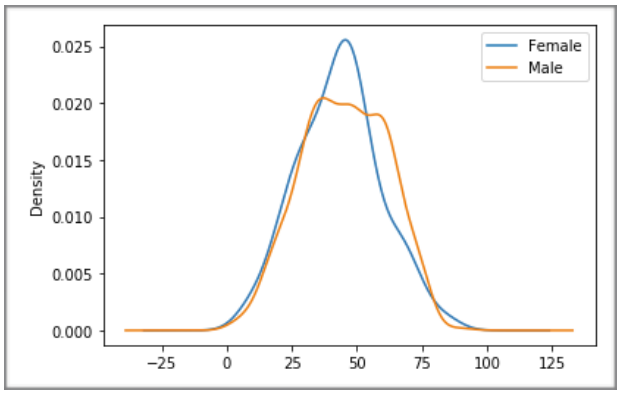
We can observe that the dataset is made up of several all numeric attributes (int or float) except one: *gender.* This feature can be “Male” or “Female” and is necessary to substitute string to int and for this reason we can use a *sklearn preprocessing* library where there is a *LabelEncoder()* function that transforms what are strings into integer.

Another analysis that can be done from a graphical and mathematical point of views is to see how the dataset is distributed numerically.
This data set contains 441 male patient records and 142 female patient records, the total of entry is egual to 583.

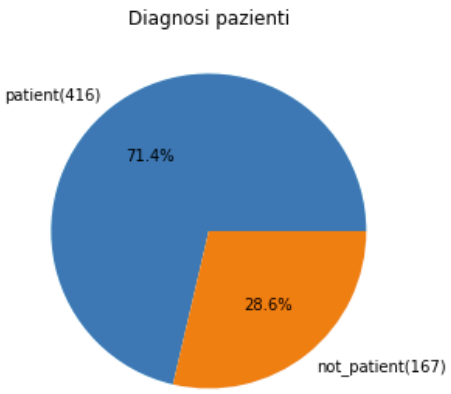


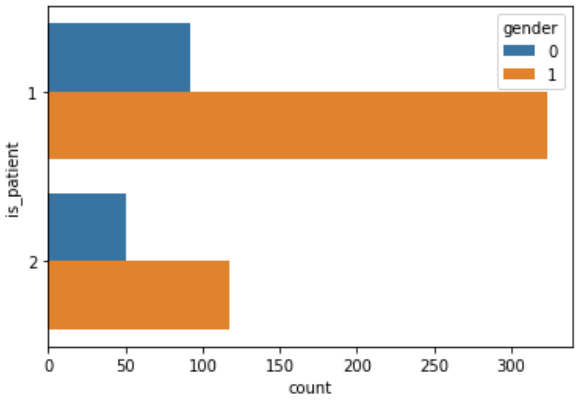


In this plot we can observe the age curve for male and female. For both gender we can observ that the highest peak is around 40-60 years old in our dataset.



A relationship that can help us well to understand how the dataset is distributed is to understand how many are characterized by an *is_patient* at 1 and how many at 2 differentiating by gender and we note very well that there is an imbalance in favor of the number 1.





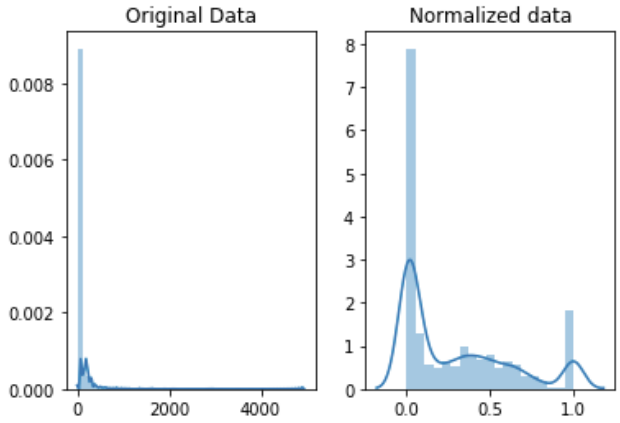
Another important activity is search eventually **missing values** (valori NaN). We can found that for *alkphos* feature there are 4 values of NaN and we replace these NaN values with 0.

1.3 Normalizzazione

As already observed previously, what is easily noticed is that the data are very different from each other: for some features we have very low values for example for '*tot_bilirubin*' we can found a minimum value egual to 0.4, a maximum value egual to 75 and a average egual to 3.29. Completely opposite, we can find the '*tot_proteins*' feature where there is a minimum value of 63, a maximum value of 2110 and an average of 63. For this reason, it is not appropriate to apply a standardization at this time, but a normalization. In this technique, data is scaled to a fixed range, typically from 0 to 1. It is a good technique to use when you don't know the distribution of data or when you know that the distribution is not Gaussian. The formula is as follows:

$$z = \frac{x - min(x)}{max(x) - min(x)}$$

In this distplot we can see how change the data before and after normalization:



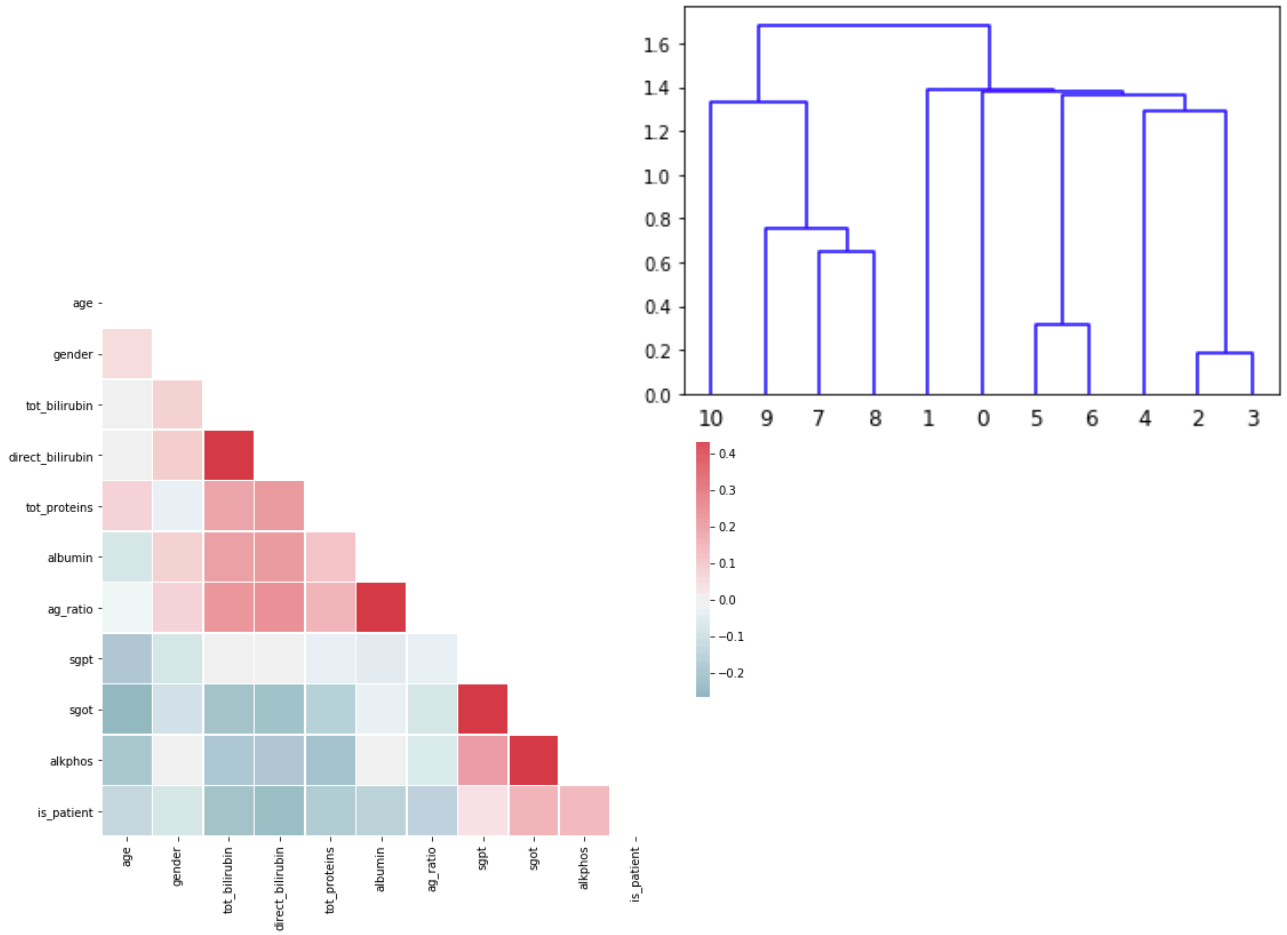
1.4 Correlazione

The next set in data analysis was to see the correlations between the different attributes and it is noted that the stronger correlations are those between:

- *tot_bilirubin* e *direct_bilirubin;*
- *ag_ratio* e *albumin;*
- *sgpt* e *sgot*

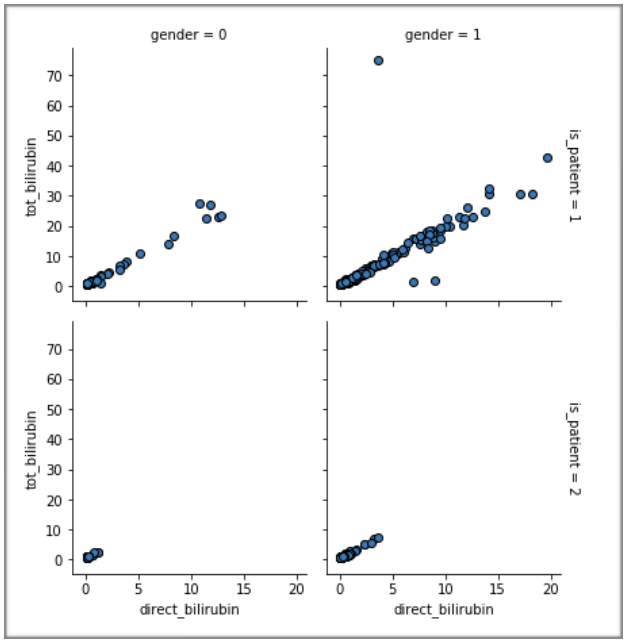
	age	gender	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	is_patient
age	1.000000	0.056560	0.011763	0.007529	0.080425	-0.086883	-0.019910	-0.187461	-0.265924	-0.203418	-0.137351
gender	0.056560	1.000000	0.089291	0.100436	-0.027496	0.082332	0.080336	-0.089121	-0.093799	0.008541	-0.082416
tot_bilirubin	0.011763	0.089291	1.000000	0.874618	0.206669	0.214065	0.237831	-0.008099	-0.222250	-0.192539	-0.220208
direct_bilirubin	0.007529	0.100436	0.874618	1.000000	0.234939	0.233894	0.257544	-0.000139	-0.228531	-0.186023	-0.246046
tot_proteins	0.080425	-0.027496	0.206669	0.234939	1.000000	0.125680	0.167196	-0.028514	-0.165453	-0.217864	-0.184866
albumin	-0.086883	0.082332	0.214065	0.233894	0.125680	1.000000	0.791966	-0.042518	-0.029742	0.004184	-0.163416
ag_ratio	-0.019910	0.080336	0.237831	0.257544	0.167196	0.791966	1.000000	-0.025645	-0.085290	-0.062993	-0.151934
sgpt	-0.187461	-0.089121	-0.008099	-0.000139	-0.028514	-0.042518	-0.025645	1.000000	0.784053	0.223185	0.035008
sgot	-0.265924	-0.093799	-0.222250	-0.228531	-0.165453	-0.029742	-0.085290	0.784053	1.000000	0.654450	0.161388
alkphos	-0.203418	0.008541	-0.192539	-0.186023	-0.217864	0.004184	-0.062993	0.223185	0.654450	1.000000	0.148265
is_patient	-0.137351	-0.082416	-0.220208	-0.246046	-0.184866	-0.163416	-0.151934	0.035008	0.161388	0.148265	1.000000

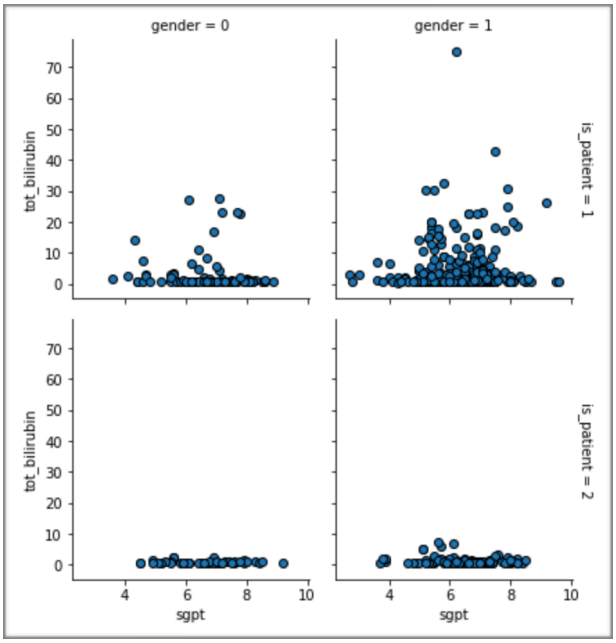
Another way to identify possible correlations between pairs of features could be to create a hierarchical clustering that receives as input a correlation matrix. In the end decide on a threshold through an elbow chart and 'cut' in a specific high. Once the cut is made, we will be able to identify which pairs are closely related to each other.



The same result could also be achieved by making scatter plots between all the combinations of attributes contained in the dataset.
For example between *tot_bilirubin* and *direct_bilirubin* (in the left graph below) and we see a directional proportionality. At the same time we can see another example between *sgpt* e *tot_bilirubin* (in the right graph below)in which can observe an inverse proportionality.

After this analysis we can delete: **direct_bilirubin, albumin and sgpt features**



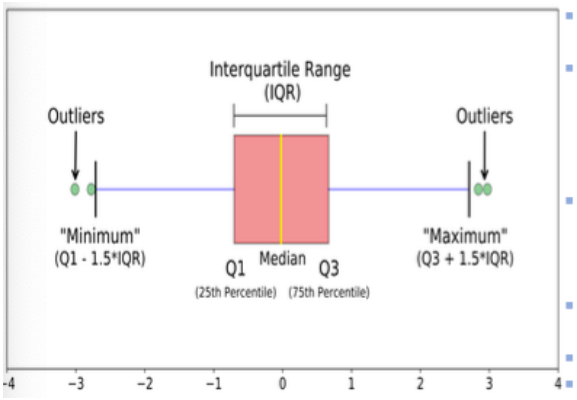


1.5 Outlier detection

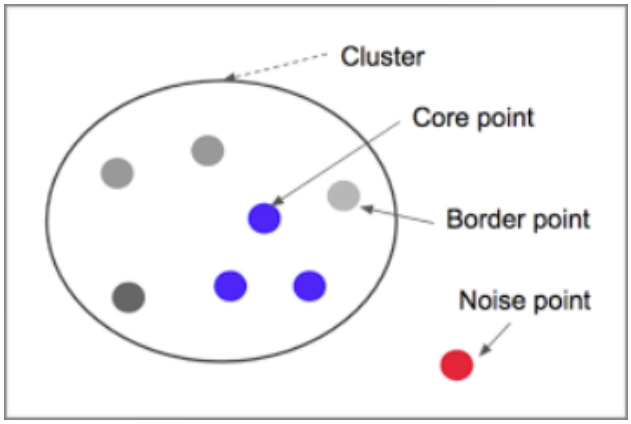
Another analysis carried out was that of the outliers, or anomalous values in a set of observations. Outliers are a data object that deviates significantly from the rest of the objects, are extreme values as if it were generated by a different mechanism. An outlier are different from noisy data. Noise is a random error or variance in a measured variable. In general, noise is not interesting in data analysis, invading outlier detection.
Outliers can also come in different flavours, depending on the environment: *point outliers*, *contextual outliers*, or *collective outliers*. Point outliers are single data points that lay from the rest of the distribution; contextual outliers can be noise in data, such as punctuation symbols when realizing text analysis or background noise signal when doing speech recognition; collective outliers can be subsets of novelties in data such as a signal that may indicate the discovery of new phenomena. In the our case the type of outlier is the first, the point outlier, because as in the case of the attribute in *ag_ratio* they are single and few points that are distant from the distribution.
In machine learning and in any quantitative discipline the quality of data is as important as the quality of a prediction or classification model.
Boxplots are a standardized way of displaying the **distribution** of data based on a five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum").

- **medium** (Q2/50th Percentile): the middle value of the attribute;
- **first quartile** (Q1/25th Percentile): the middle number between the smallest number (not the "minimum") and the median of the attribute;
- **third quartile** (Q3/75th Percentile): the middle value between the median and the highest value (not the "maximum") of the attribute;
- **interquartile range (IQR)**: 25th to the 75th percentile;
- **whiskers** (shiwn in blue)
- **outliers** (shown as green circles)

Outliers may be plotted as individual points.



Il seguente grafico indica il box plot di tutti i pazienti con patologia 1 (**ispatient=1**):
Il seguente grafico indica il box plot di tutti i pazienti con patologia 2 (**ispatient=2**):
Altro algoritmo utilizzabile per l'outliers detection è l'algoritmo di DBSCAN. **DBSCAN** is a **density-based clustering** non parametric algorithm: given a set of points in some space, it **groups together** points that are closely packed together (points with manu nearby neighbors), marking as **outliers** points that lie one in low-density regions (whose nearest neighbors are too far away).



Once the outlier has been identified, two ways are possible: **correct** or **remove** the outlier. Choosing one of the two is very difficult and there are many factors to consider when making a decision. Since the dataset is relatively small and also some values such as outliers could be fundamental for the purpose of prediction it is advisable not to proceed with some outliers removal.

2. Feature Extraction

The objective of this chapter is to apply **feature extraction** techniques useful for reducing the size of the dataset. There are several techniques for doing this and below we will first illustrate the theoretical concepts then the practical effects on the dataset in question.

2.1 Theory

Feature Extraction aims to reduce the number of features in a dataset creating new features from the existing ones (and then discarding the original features). These new reduced set of features should then be able to summarize most of the information contained in the original set of features. In this way, a summarized version of the original features can be created from a combination of the original set. Another commonly used technique to reduce the number of feature in a dataset is Feature Selection. The difference between Feature Selection and Feature Extraction is that feature selection aims instead to rank the importance of the existing features in the dataset and discard less important ones (no new features are created).
Noi useremo la feature extraction ed in particolare analizzeremo due tecniche che sono: PCA e LDA.

2.2 Standardization

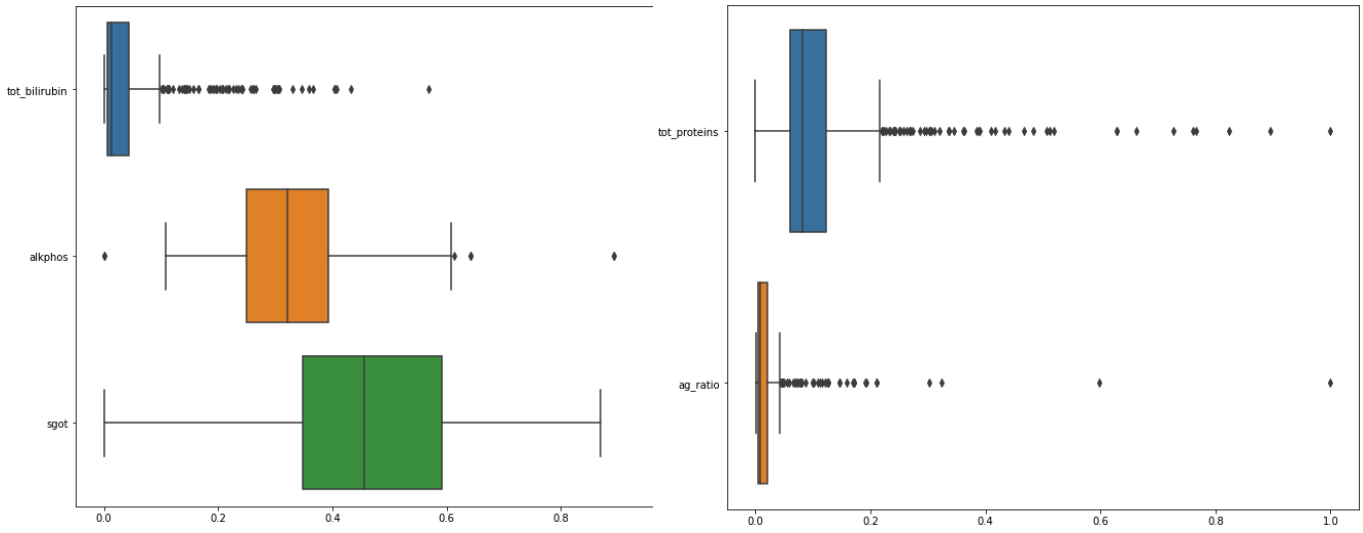
The terms normalization and standardization are sometimes used interchangeably, but they usually refer to different things. Normalization usually means to scale a variable to have a values between 0 and 1, while standardization transforms data to have a mean of zero and a standard deviation of 1. This standardization is called a z-score, and data point can be standardized with the following formula:

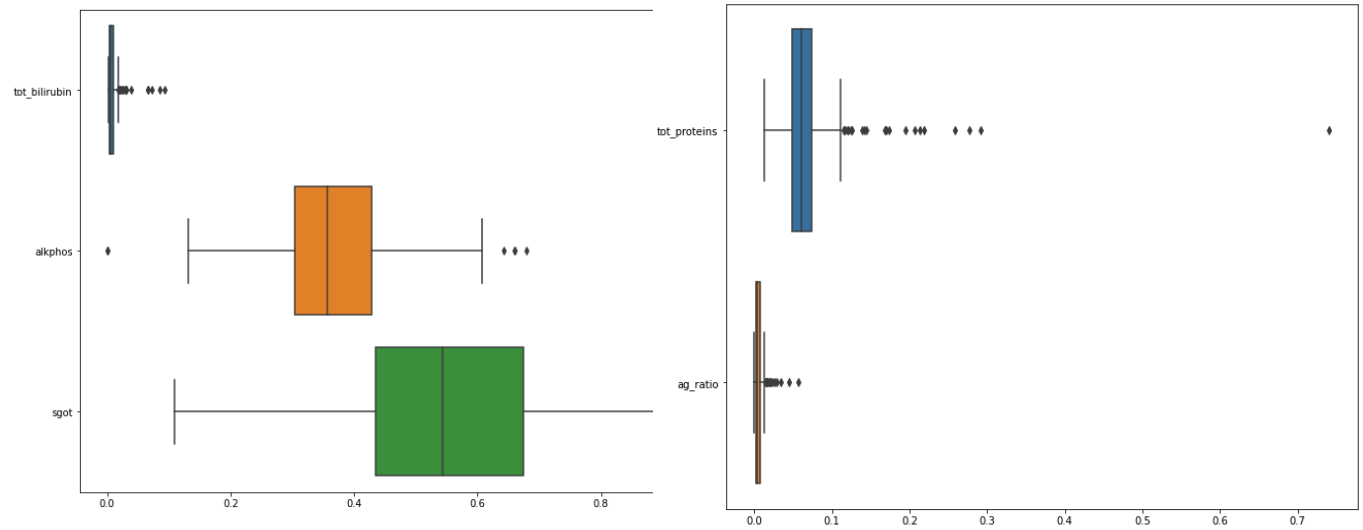
$$z_i = \frac{x_i - \bar{x}}{s}$$

Is necessary to scaling data before performing PCA. The PCA calculates a new projection of the data. The new axis are based on the standard deviation of your variables, so a variable with a high standard deviation will have a higher weight for the calculation of axis than a variable with a low standard deviation. If you normalize your data, all variables have the same standard deviation.

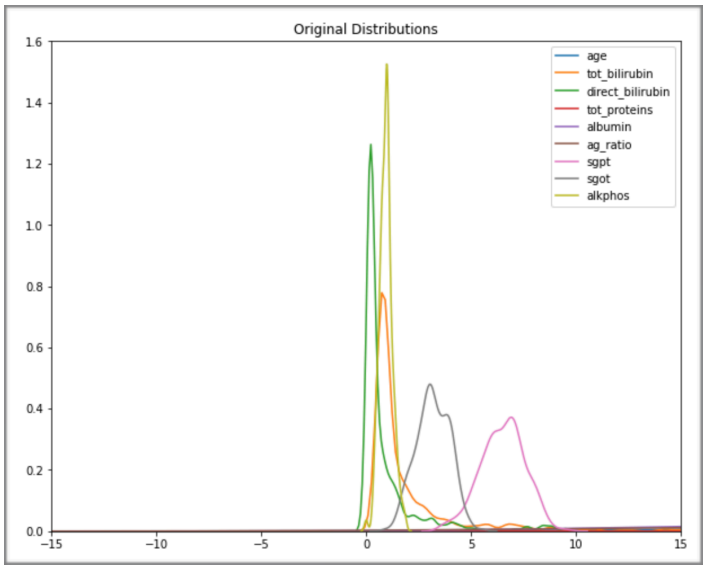
Normalization is important in PCA since it is a variance maximizing exercise. It projects your original data onto directions which maximize the variance.
If we didn't normalize the data we would have some axes where the variance would be greater and other axes where we would have a minor variance. To standardize the data and to then be able to calculate a PCA without making mistakes, it is advisable to scale the data. If we normalize the data, all variables have the same standard deviation, thus all variables have the same weight and the PCA calculates relevant axis.

In this project we use **StandardScaler** from *sklearn.preprocessing* standardizes a feature by subtracting the mean and the scaling to unit variance. Unite variance means dividing all the values by the standard deviation. StandardScaler results in a distribution with a standard deviation equal

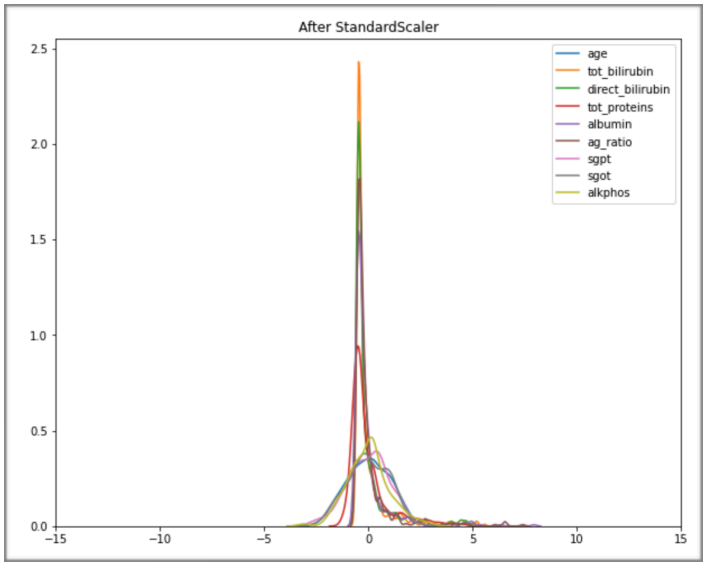




to 1. The variance is equal to 1 also,
because variance is equal to standard deviation squared (and 1 square is equal to 1).
Below we can see the graph before applying standardization with StandardScaler:



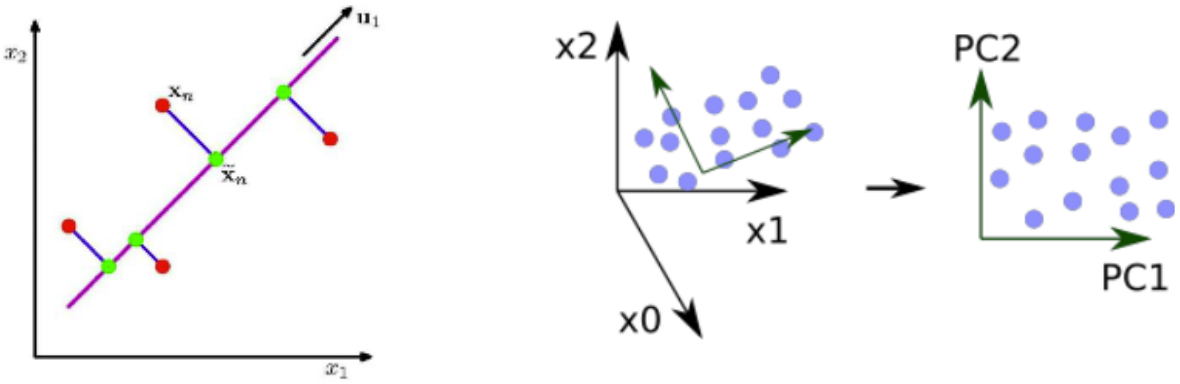
Below we can see the graph after applying standardization with StandardScaler:



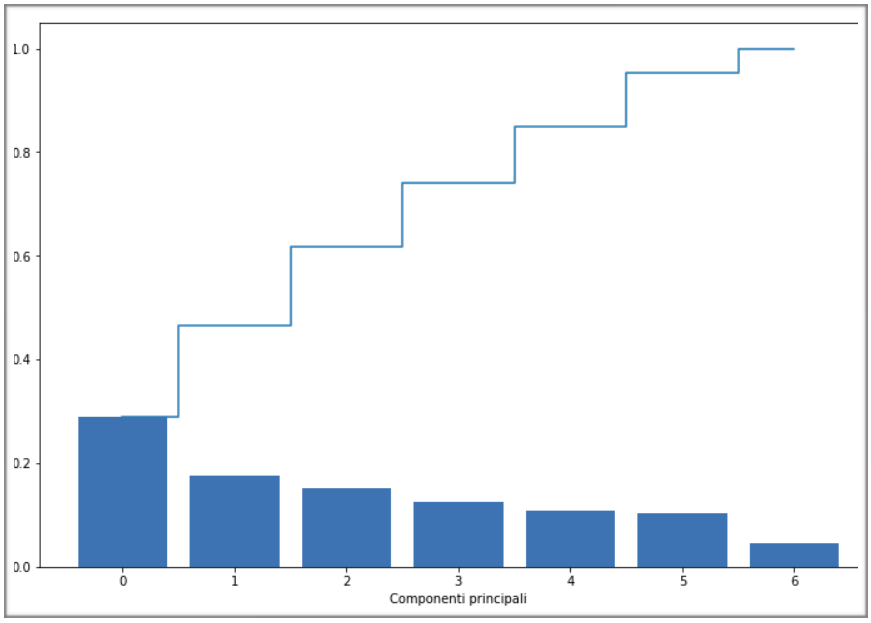
2.3 Principle Components Analysis (PCA)

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same analytical results.
Data reduction strategies include *dimensionality reduction*, *numerosity reduction*, and *data compression*.
Dimensionality reduction is the process of reducing the number of random variables or attributes under consideration. Dimensionality reduction methods include *principal components analysis (PCA),* which transform or project the original data into a smaller space.
Principal Component Analysis (PCA) is an unsupervised linear transformation technique that is widely used across different fields, most prominently for feature extraction and dimensionality reduction. The Principale Component analysis, in maths, is a orthogonal projection of the data onto a lower dimension linear space that:

- **maximizes** variance of projected data (purple line);
- **minimizes** mean squared distance between data point and projections (sum of blue lines);



Given data points in a d-dimensional space, project the into a **lower dimensional** space while **preserving as much information** as possible. In particular, choose projection that **minimizes squared error** in reconstructing the original data.
Using a specific python library (sklearn decomposition) we can plot a bar chart that says what is the variance of our attributes.



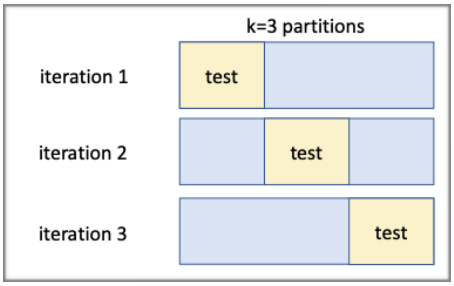
The variance values are:
[0.2903902 0.17530184 0.15084776 0.12498452 0.10872371 0.10345407, 0.0462979].
We see that the first six components come to have a good percentage of the total, so we are going to do the PCA on a number of components equal to 6. So we come to have a reduced dimensionality compared to the problem of origin equal to 6 compared to the initial 10.

3. Metrica di valutazione

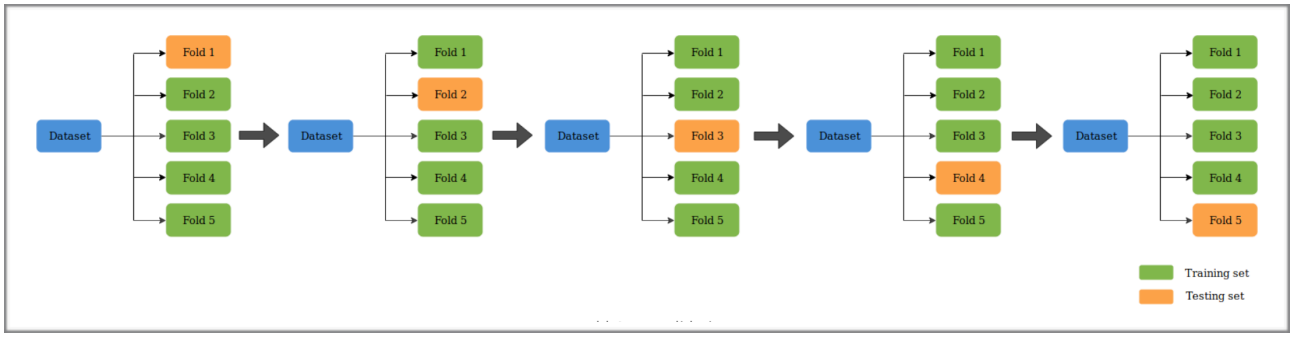
In this chapter we focus on data analysis with some classification algorithms and we will evaluate their performance by further analysis in the next chapter.

3.1 Primo step: K-fold cross validation

Usually, the first step is split the data set into training and testing sets. Un'altra tecnica molto usata è dividere il data set non in due parti, bensì in 3 parti: training set, validation set and test set. In questo modo, nel validation set noi possiamo testare il modello, aggiustarlo modificando alcuni parametri dell'algoritmo e poi infine, usarlo nel test.
This method in not very reliable as the accuracy obtained for one test set can be very different to the accuracy obtained for a different test set. K-fold Cross Validation (CV) provides a solution to this problem by dividing the data into k folds. At each iteration select a partition to be used as test set and the others will be the training set.

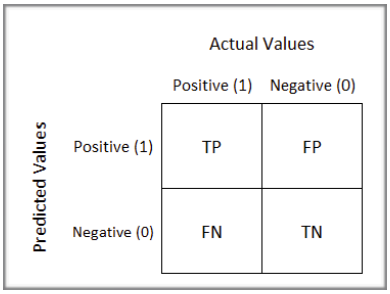


K-fold Is where a given data set is split into K number of sections/folds where each fold is used as a tasting set at some point. In the image above the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.



3.2 Metrics to evaluate classifier algorithm

3.2.1 Confusion Matrix



It is a performance measurement for machine learning classification problem. It is a table with 4 different combinations of predicted and actual values.
TP is True Positive and means predicted positive and it's true; TN is True Negative and means "predicted negative and it's true"; FP is False Positive and means "predicted negative and it's true"; FN is False Negative and means "predicted negative and it's false".
Now we want understand how calculate other important measurement for example Recall, Precision and F-measure.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$Recall = \frac{TP}{TP + FN}$ as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized.

$Precision = \frac{TP}{TP + FP}$ get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High precision indicates an example labelled as positive is indeed positive.

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}.$$

3.2.2 F1 Score

F1 Score is the harmonic mean between precision and recall. The range for F1 Score is [0,1]. This measure means how precise the classifier is (how many instances it classifies correctly), as well as how robust it is.

Mathematically, it can be expressed as: $F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$

3.2.3 Log Loss

Log loss works penalizing the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples. Suppose, there are N samples belonging to M classes, the the Log Loss is calculated as below:

$$LogLoss = \frac{-1}{N} \sum_{i=0}^{N} \sum_{j=1}^{M} y_{ij} * log(p_{ij})$$

Dove y_{ij} indicates whether sample I belongs to class j or not;

p_{ij} indicates the probability of sample I belonging to class j.

Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy. In general, minimizing Log Loss gives greater accuracy for the classifier.

4 Classificazione

We can predict in advance which classification algorithms are best suited to our problem. Among these we can mention the algorithm of **Logistic Regression** because performs binary classification, so the label outputs are binary. It takes linear combination of features and applies non-linear function (sigmoid) to it. The second algorithm that we analyze is **SVM** because is used in pattern recognition and classification problems when the data has exactly two classes. Another algorithm that works very well is **LDA, random forest classifier, decision tree** and in finally **k-NN.**

What will always be done, for each classification algorithm, will be to find the best hyperparameters using the GridSearchCV function with k-Fold where the number of splits is 10.

4.1 Logistic Regression

Logistic Regression is used when the dependent variable (target) is categorical and in this case we have a categorical problem because we have only two targets: 1 or 2.
The straight line dividing the points is given by the following equation: $b + w_1 x_1 + w_2 x_2 = 0$ where b and w are **bias** and **pesi** in our model. Once weights and biases are found we can calculate our prediction $z = b + w_1 x_1 + w_2 x_2$ and the z value can have a value greater or less than 0 depending on whether it is above the straight line or below. Furthermore z is a contiguous variable while we need a categorical variable to know in which class a group belongs. To do this we must use an **activation function** which takes the z value returned by the function and returns the corresponding class.
The activation function is the **sigmoidal** function that corresponds to $P(Y = 1) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$ where e is the number of Euler (2.79). The sigmoidal function enjoys the interesting property of reducing real numbers in a range between 0 and 1. The value of the sigmoidal function for z = 0 corresponds to 0.5.
So in a nutshell, logistic regression works like this protocol:

- we calculate z ($-\infty < z < \infty$);
- use a activation function on z;

- $0 \leq \hat{y} \leq 1$ indicates the probability that the case in question belongs to a positive class.

Through some mathematical manipulations it is obtained:

$\frac{P(Y=1)}{1-P(Y=1)} = e^{\beta_0+\beta_1 x}$ and if we apply the log we can obtain:

$log(\frac{P(Y=1)}{1-P(Y=1)}) = \beta_0 + \beta_1 x$. This monotone transformation is called the **log odds** or **logit** transformation of $P(Y=1) = P(x)$

Training is done in the same way as any regression model. The only thing that changes is the cost function, in this case is better to use affinity, not cost. The most used is the likelihood that corresponds to the question "given the property X and the weights W, what is the probability of obtaining the correct target y? Being a probability the target will be between 0 and 1. For a numerical question, during the training phase it is better to use the log likelihood $l(W) = logL(W)$ which corresponds to the logarithm of likelihood. During the training phase it is necessary to maximize the log likelihood through an optimization algorithm called gradient ascent that follows the logic of the descent with the difference that it seeks the maximum point instead of the minimum one, or you can use the gradient descent as we have seen to minimize the negative of log likelihood.

In the GridSearch we put this parameters:

Penalty	l1, l2
C	np.logspace(0,4,10)

Best hyperparameters:
- C = 2.7825
- Penalty = l2

Accuracy 0.69
Classification Report:

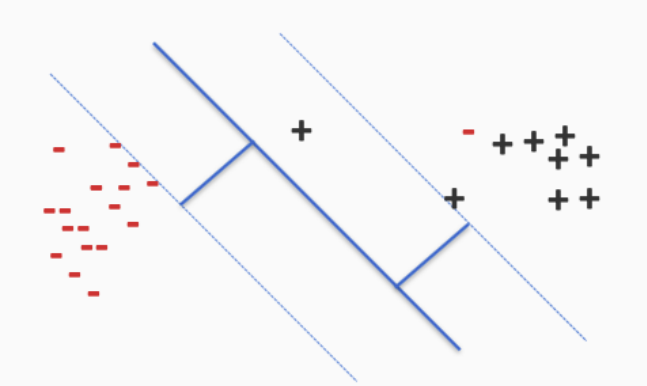
	Precision	Recall	F1-score	Support
is_patient = 0	0.72	0.87	0.79	78
is_patient = 1	0.57	0.33	0.42	39
Accuracy			0.69	117
Macro avg	0.64	0.60	0.61	117
Weighted avg	0.67	0.69	0.67	117

Class 1 is predicted much better than class 2. This is justified by the fact that the dataset, as already stated above, appears to be slightly unbalanced as there are more patients in class 1 instead of class 2.

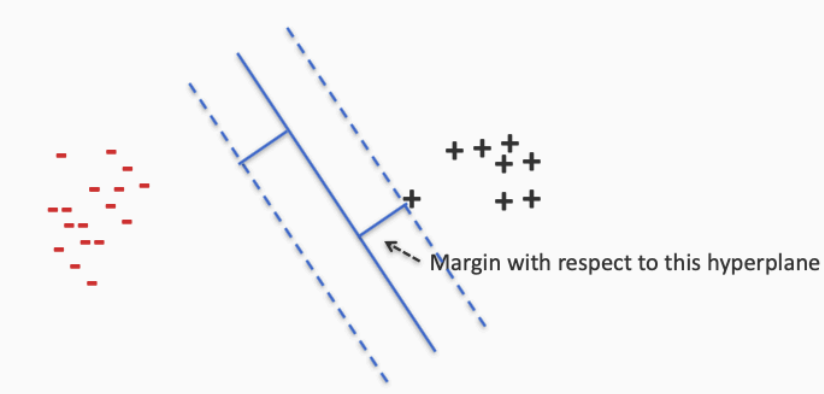
4.2 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning technique that is widely used in pattern recognition and classification problems - when your data has exactly two classes.

The techniques based on the Support Vector Machine are based on the concept of hyperplane: the aim is to build a class of learning machines that build a hyperplane capable of separating, as far as possible, the two classes in question.
The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



The dimension of the hyperplane depends upon the number of features. If the number od input features is 2, then the hyperplane is just a line. If the number of input features is 3, the the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.
For the first step we can try to find the linear separator that maximizes the margin.
In general the equation for a hyperplane has the form



$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$. The vector $\beta = (\beta_1, \beta_2, \ldots, \beta_p)$ is called the normale vector. Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes:

- $max_{\beta_0,\beta_1,\ldots,\beta_p} M$
- subject to $\sum_{j=1}^{p} \beta_j^2 = 1$
- $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$ for all $i = 1,\ldots,N$

Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier. The support vector classifier maximizer a soft margin.
The addition of an additional point could cause a radical change of the classifier. The key idea is a creation of the support vector classifier to allow soma examples to “break into the margin”. The problem of maximizes margin becomes:

- $max_{\beta_0,\beta_1,\ldots,\beta_p,\varepsilon_1,\ldots,\varepsilon_n} M$
- Subject to $\sum_{j=1}^{p} \beta_j^2 = 1,$
- $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i), \varepsilon \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C$

To map data into a higher dimension, a new property must be built and this process is extremely expensive in terms of computing resources especially if our dataset has many properties. Here comes the kernel trick which allows us to obtain a similar result without going to operate in a larger dimensional space by simply using a kernel function. A kernel function is basically a metric of the similarity between two examples: there are several and the most used is the **Kernel Gaussiano (o Radial Basis):**

$K(x,l) = e^{\frac{-||x-l||^2}{2\sigma^2}}$ when two examples are very different then this function will tend to zero. When they are very similar, however, it tends to 1. The constant sigma must be chosen by us and controls the sensitivity of the difference between two examples. With a smaller sigma value a greater difference of the examples will be more penalized and therefore will tend to zero more easily.
We have also **kernel lineare** which corresponds to a simple application of the SVM and allows to obtain linear decision boundaries; then we have the Gaussian kernel just seen that it is generic and to be used if you don't know how to move; other 2 kernel are **sigmoidal** and **polynomial.**

In the GridSearch we put this parameters:

Kernel	*Linear/RBF*
C	*[1,10,100,1000]*
Gamma	*[1e-3, 1e-4]*

Best hyperparameters:

- Kernel: RBF
- Gamma = 0.001
- C: 1

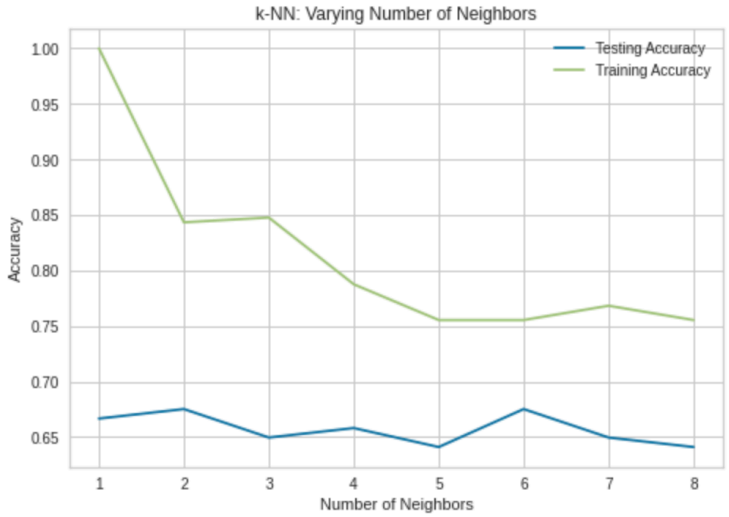
Accuracy Score = 0.67

Classification Report

	Precision	Recall	F1-score	Support
is_patient = 0	0.67	1.00	0.80	78
is_patient = 1	0.00	0.00	0.00	39
Accuracy			0.67	117
Macro avg	0.33	0.50	0.40	117
Weighted avg	0.44	0.67	0.53	117

4.3 k-Nearest Neighbors

The K-Nearest Neighbors classification algorithm is based on the concept of classifying an unknown sample considering the class of k samples closest to the training set. The new sample will be assigned the label of the class to which most of the nearest k samples belong. This is an incremental algorithm and is a training set model. Among the disadvantages we note that there are the fact that it is very susceptible to the presence of outliers. The value of K is very important and decisive: if too small it can be sensitive to noise, if too large it can be an expensive operation. Below you can see how the accuracy of the classifier varies according to the number of k chosen. The best k is 2 and 6.



Accuracy on test set is 0.675

	Precision	Recall	F1-score	Support
is_patient = 0	0.68	0.99	0.80	78
is_patient = 1	0.67	0.05	0.10	39
Accuracy			0.68	117
Macro avg	0.67	0.52	0.45	117
Weighted avg	0.67	0.68	0.57	117

4.4 Random Forest Classifier

Random Forest is an ensemble of decision trees. It can solve both regression and classification problems with large data sets. It also helps identify most significant variables from thousands of input variables. Random Forest is highly scalable to any number of dimensions and has generally quite acceptable performances. The finally, there are genetic algorithms, which scale admirably well to any dimension and any data, with minimal knowledge of the data itself, with the most minimal and simplest implementation being the microbial genetic algorithm. With Random Forest however, learning may be slow (depending on the parameterization) and it is not possible to iteratively improve the generated models. Random Forest can be used in real- world application such as:

- Predict patient for high risks;
- Predict parts failures in manufacturing
- Predict loan defaulters.

Using **ensemble** learning improves generalization of unknown data and reduces the risk of overfitting.
The construction of a random forest follows the following steps:

1. choose the number of trees that will make up the forest;
2. we take a subset of data, randomly, from the dataset, and use it to create a decision tree;
3. train a tree on this subset;
4. we repeat steps 2 and 3 until we have built all the trees that we set out to create;

To perform a classification, we use all the trees and then we average the various predictions. The big advantage of this random forest is that it has a minimum number of hyperparameters, in fact the only parameter we need to worry about is the number of trees that will form the forest, it is usually 10, but increasing it could get a better model.

In the GridSearch we put this parameters:

Criterion	Gini - Entropy
Max_depth	[5,6,7,8,9,10,11,12]
Max_features	[1,2,3]
n_estimator	[14,15,16,17,18,19]

Accuracy on train set 0.7511
Accuracy on test set 0.6667.

4.5 Decision Tree

A Decision Tree is a Supervised Machine Learning where the data is continuously split according to a certain parameter.
Decision Tree consists of:
- **nodes**: test for the value of a certain attribute;
- **edge**: correspond to the outcome of a test and connect to the next node or leaf;
- **leaf nodes**: terminal nodes that predict the outcome

There are two main types of Decision Trees:
1. Classification Trees (for categorical target)
2. Regression Trees (for continuous target)

We use the classification trees In this project and a key idea is to use a decision tree to partition the data space into cluster regions and empty regions.
In Decision Tree classification a new example is classified by submitting it to a series of tests that determine the class label of the example. These tests are organized in a hierarchical structure called a decision tree.

The algorithm splits the data into subsets, which are then split repeatedly into event smaller subsets until the process stops when the algorithm determines the data within the subset are sufficiently homogenous.

We start at the tree root and split the data on the feature that results in the largest information gain (IG). In an iterative process, we can then repeat this splitting procedure at each child node until the leaves are pure. This means that the samples at each leaf node all belong to the same class. We may set a limit on the depth of the tree to prevent overfitting.

The metric that measures how good the gain is called impurity. The goal is to get to the leaf by trying to ask as few questions as possible.
There are other metrics to calculate impurities, the most popular are: **gini** and **entropy**. The first is less expensive for the calculation mode:

$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$ The Gini index measure total variance across the K classes. The Gini index takes on a small value if all of the p_{mk}'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity - a small value indicates that a node contains predominantly observations from a single class.
The second one is cross-entropy:

$D = -\sum_{k=1}^{K} p_{mk} log p_{mk}$ The mini index and the cross-entropy are very similar numerically.

Advantages of Trees:

- Trees are very easy to explain to people;
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous;
- Trees can be displayed graphically, and are easily interpreted even by a non-expert;
- Trees can easily handle qualitative predictors without the need to create dummy variables;

Disadvantages of Trees:

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen previously.

In the GridSearch we put this parameters:

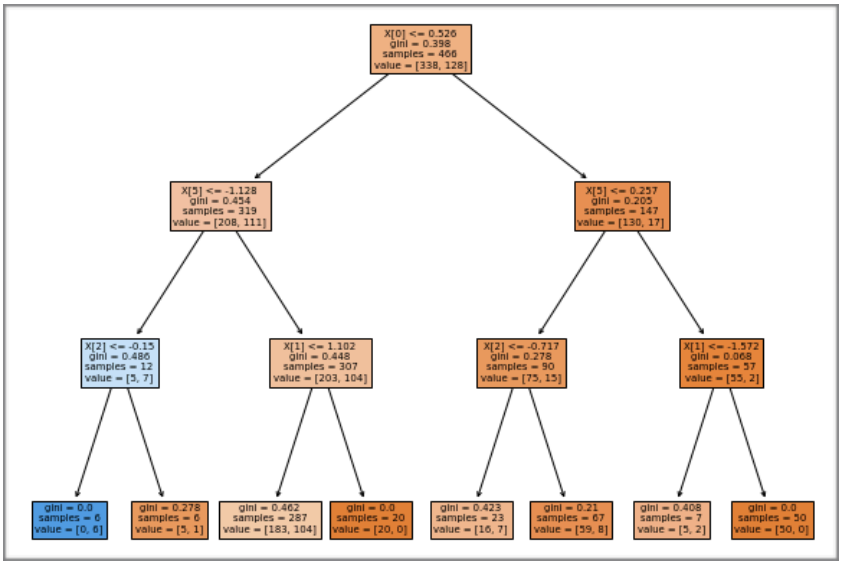
max_features	auto','sqrt','log2'
max_depth	[1,2,3,4,5,6,7]
min_samples_leaf	[1,2]
min_samples_split	[2,3,4,5,6,7,8,9,10,11,12,3,14,15]

The hyper parameters are:

- max_features = 'auto'
- min_samples_leaf=1
- min_samples_split=5
- max_depth=3
- random_state = 12

The accuracy on test set: 0.67

	Precision	Recall	F1-score	Support
is_patient = 0	0.67	1.00	0.80	78
is_patient = 1	0.00	0.00	0.00	39
Accuracy			0.67	117
Macro avg	0.33	0.50	0.40	117
Weighted avg	0.44	0.67	0.53	117



5. Ulteriori analisi

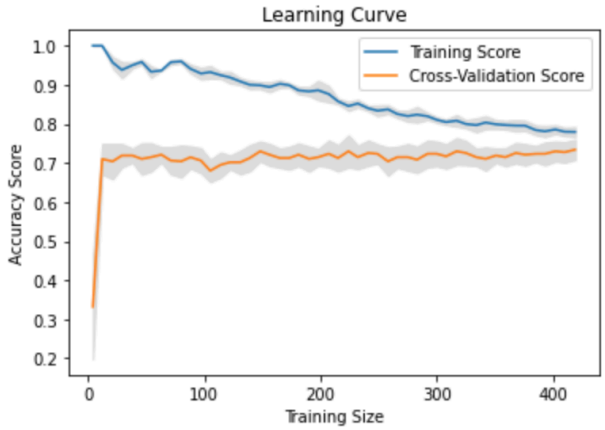
In questo capitolo ci soffermeremo a analizzare in maniera più dettagliata alcuni aspetti del nostro dataset che nelle precedenti analisi non sono state messe in risalto.

I risultati dei modelli predittivi sono stati:

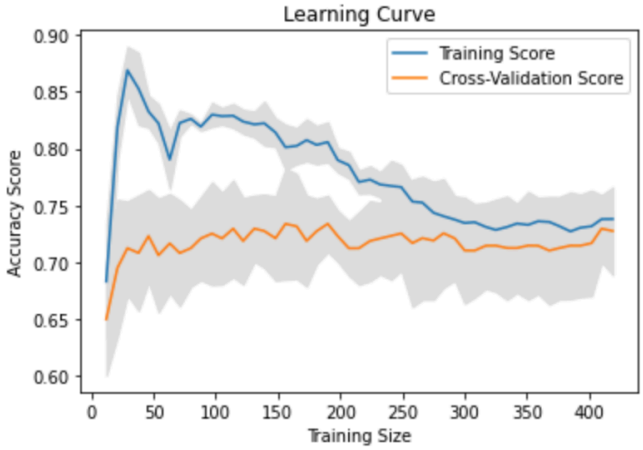
Algoritmo di classificazione	Accuratezza
Logistic Regression	0.69
k-NN	0.67
RBF-SVC	0.67
Random Forest	0.67
Decision Tree	0.67

5.1 Learning Curve

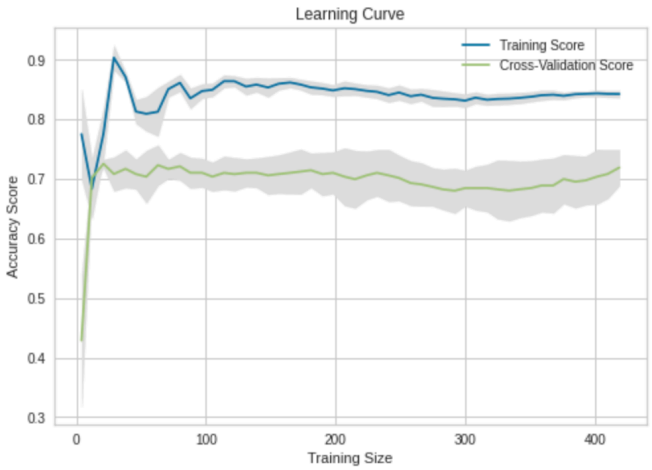
A Learning curve is a plot that shows time or experience on the x-axis and learning or improvement on the y-axis. The model can be evaluated on the training dataset and on a hold out validation dataset after each update during training and plots of the measured performance can created to show learning curves.
The shape and dynamics of a learning curve can be used to diagnose the behavior of a machine learning model and in turn perhaps suggest at the type of configuration changes that may be made to improve learning.
We can observe three dynamics:
- Underfit
- Overfit



This plot is referred to Random Forest Classifier. Is very similar to the previous one. Probably with the increase of the data you can get faster to an overfit situation. The other difference is that the training score decreases faster

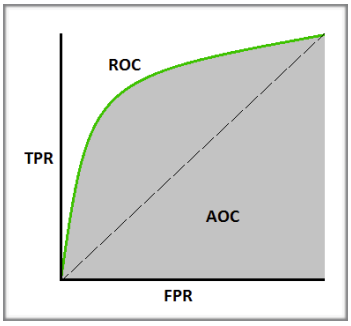


The graph in question concerns the Logistic Regression classifier. We observe that it already works very well when it has less than 200 data. Accuracy with increasing data seems to increase, perhaps with a greater supply of data we would have had a better performance



This plot represents the k-NN algorithm. Is an unrepresentative validation dataset means that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize. In this case, it indicates that the training dataset may be easier for the model to predict than the test dataset.

5.2 ROC curve



ROC (**Receiver Operating Characteristics**) curve is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (**Area Under the Receiver Operating Characteristics**). ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0 and 1. True Positive Rate (TPR) or Recall is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.
The FPR (False Positive Rate) is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives.

$$TruePositiveRate(TPR) = \frac{TP}{TP + FN}$$

$$FalsePositiveRate(FPR) = \frac{FP}{FP + TN}$$

Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives. Larger values on the y-axis of the plot indicate higher true positives and lower false negatives. A model with perfect skill is represented at a point (0,1). A model with perfect skill is represented by a line that travels from the bottom left of the plot to the top left and then across the top to the top right. The value of AUC is 0.74 and is very good, it means that the class are well separated.

$\text{ROC of class } 0, \text{AUC} = 0.73$
$\text{ROC of class } 1, \text{AUC} = 0.73$
$\text{Micro-average ROC curve, AUC} = 0.77$
$\text{Macro-average ROC curve, AUC} = 0.74$